



US007080325B2

(12) **United States Patent**
Treibach-Heck et al.

(10) **Patent No.:** **US 7,080,325 B2**

(45) **Date of Patent:** **Jul. 18, 2006**

(54) **GRAPHICAL DEVICE FOR
COMPREHENSIVE VIEWING AND INPUT
OF VARIABLE DATA VIA A
BROWSER-BASED DISPLAY**

2003/0055699 A1* 3/2003 O'Connor 705/7

OTHER PUBLICATIONS

Microsoft Internet Explorer, 1995-2001, Version 6.0.2800.1106CO, p. 1-2.*

Microsoft Outlook, 2000, Version 9.0.0.4527, p. 1-2.*

* cited by examiner

Primary Examiner—Ba Huynh

Assistant Examiner—Mylinh Tran

(74) *Attorney, Agent, or Firm*—Jeffrey Pearce

(75) Inventors: **Varda Treibach-Heck**, Redwood City, CA (US); **Byron G. Caldwell**, Los Angeles, CA (US); **Bruce Johnson**, Chatsworth, CA (US)

(73) Assignee: **Call-Tell LLC**, Redwood City, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 423 days.

(57) **ABSTRACT**

A graphical input device, preferably in a browser-based display, resembles a pull-down menu, with a displayed data entry field. The set of possible entries to be displayed along with the graphical input device varies depending on the user. The minimum display width needed for full-width display of the entries is evaluated by scripting and the actual display of the set is generated accordingly. The drop-down list and its listed entries may be wider than the data entry field. The user can thus view and select customized possible entries that cannot fit within the displayed data entry field. According to another aspect of the preferred embodiment of the invention, when the user is making selections from the drop-down list, multiple keyboard entries can be used to cause rapid scrolling to entries that match not only the first character entered, but also to entries whose first n characters match n characters entered while viewing the list. For hierarchically structured and itemized entry lists, matching is preferably performed first to the right of at least one delimiter. The various novel aspects of the invention are implemented using scripting embedded within HTML code so that they are available to users who are browsing web content.

(21) Appl. No.: **10/080,472**

(22) Filed: **Feb. 22, 2002**

(65) **Prior Publication Data**

US 2003/0160828 A1 Aug. 28, 2003

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **715/780**; 715/689; 715/843; 715/739; 715/783

(58) **Field of Classification Search** 345/689, 345/780, 843, 739, 783, 553

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,556,218 B1* 4/2003 Alcorn 345/749

6,760,720 B1* 7/2004 De Bellis 707/3

6,765,595 B1* 7/2004 Lee et al. 345/780

2 Claims, 3 Drawing Sheets

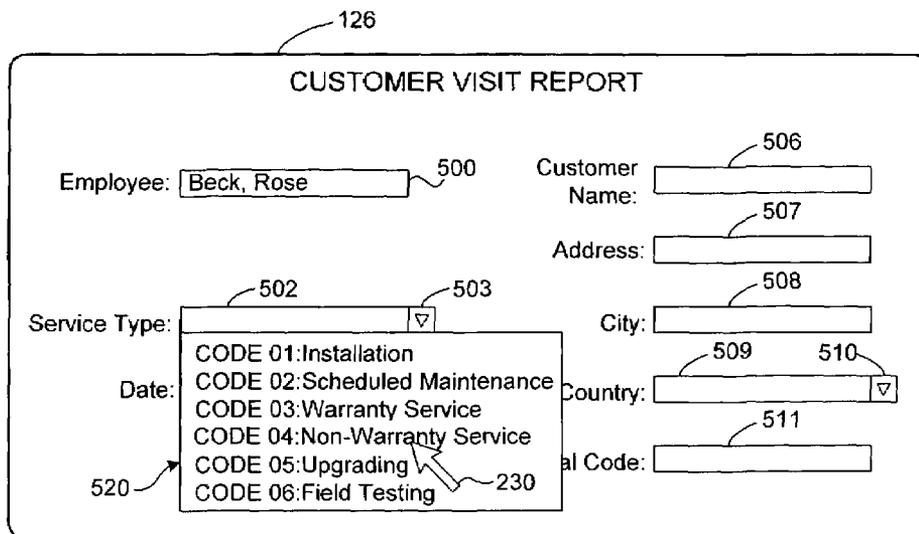


FIG. 1 (Prior Art)

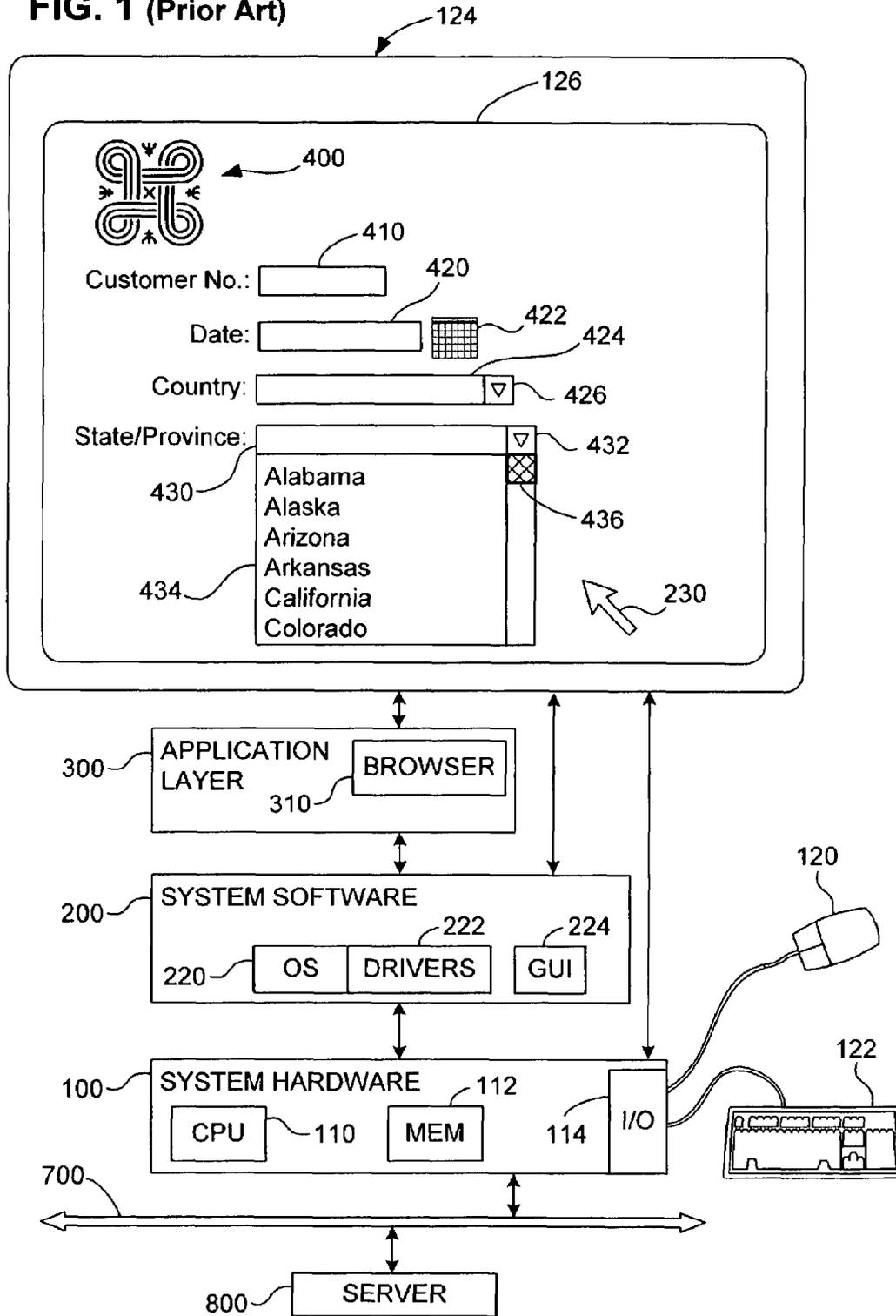


FIG. 2A

126

CUSTOMER VISIT REPORT

Employee: 500

Customer Name: 506

Address: 507

Service Type: 502 503

City: 508

Date: 504 230 505

State/Country: 509 510

ZIP/ Postal Code: 511

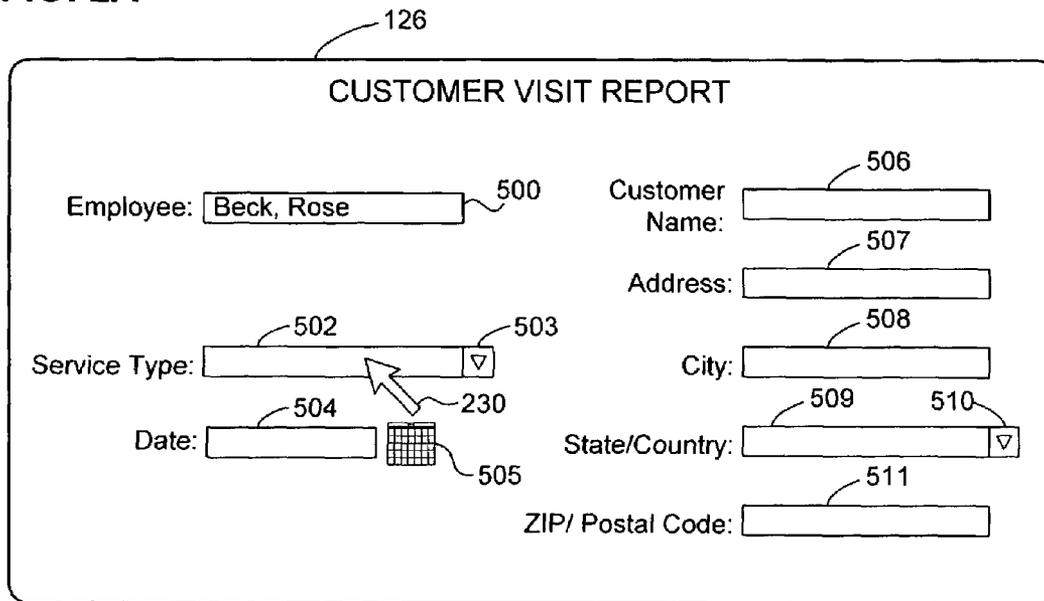


FIG. 2B

126

CUSTOMER VISIT REPORT

Employee: 500

Customer Name: 506

Address: 507

Service Type: 502 503

City: 508

Date: 520 230 505

Country: 509 510

al Code: 511

CODE 01: Installation
CODE 02: Scheduled Maintenance
CODE 03: Warranty Service
CODE 04: Non-Warranty Service
CODE 05: Upgrading
CODE 06: Field Testing

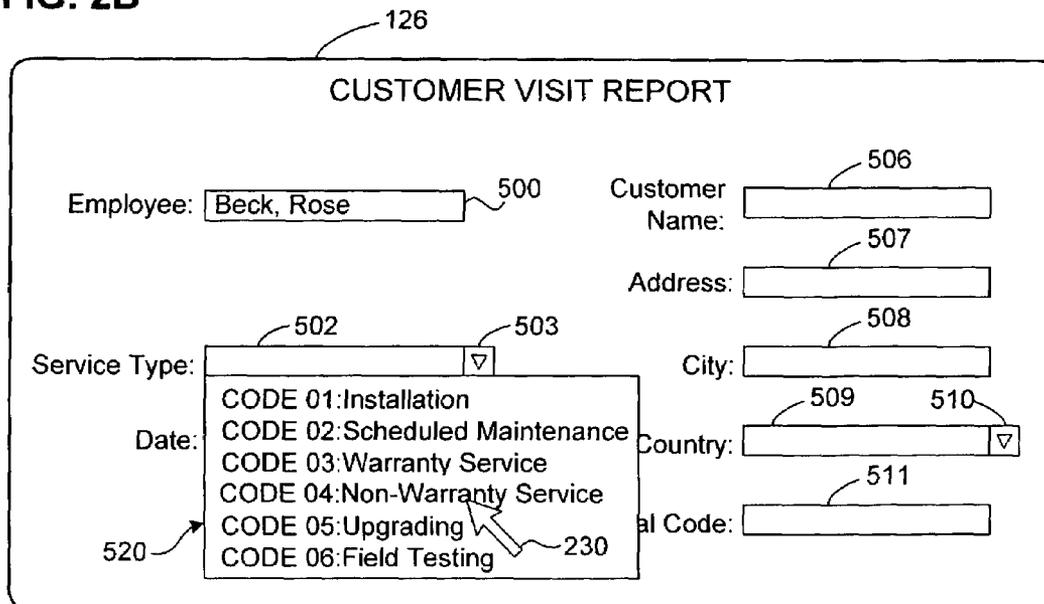


FIG. 2C

126

CUSTOMER VISIT REPORT

Employee: Beck, Rose 500

Customer Name: 506

Address: 507

City: 508

State/Country: 509 510

ZIP/ Postal Code: 511

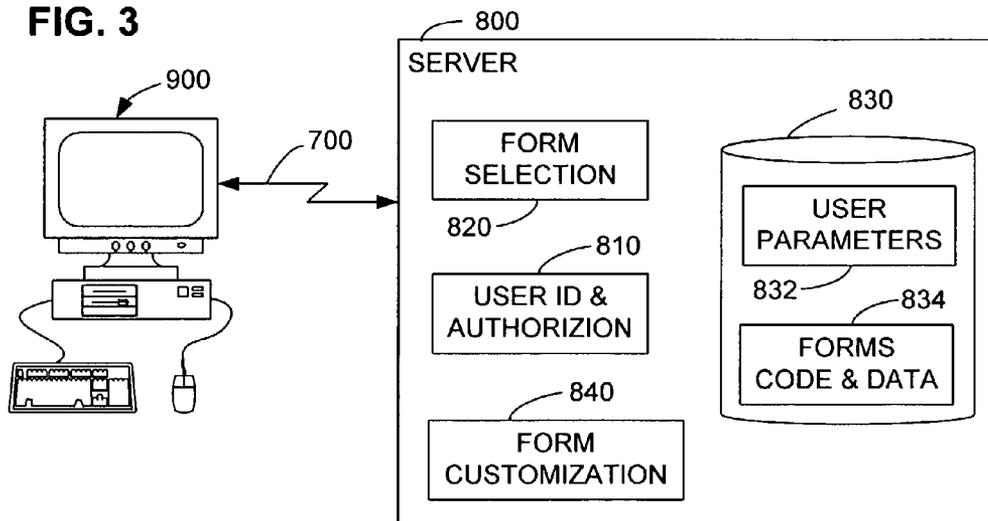
Service Type: CODE 04: Non-Warr 502 503

Date: 504

230

505

FIG. 3



**GRAPHICAL DEVICE FOR
COMPREHENSIVE VIEWING AND INPUT
OF VARIABLE DATA VIA A
BROWSER-BASED DISPLAY**

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates in general to graphical input devices and in particular to input devices in which users select from a list of choices.

2. Description of the Related Art

Sending and receiving information via networks, especially the Internet, has become a daily event for a large number of people throughout the industrialized world. Air-line tickets are ordered, books are bought, reports are submitted, subscriptions are entered, and thousands of other tasks are now routinely completed on-line.

Although the hardware devices—servers, routers, cables, etc.—that make the Internet possible are of course indispensable, they are transparent to most users. What is by definition very obvious to users, however, is the interface between them and the hardware structure. In the context of on-line information exchange, the most important components of this interface are the graphical user interface (GUI) of the user's computer in general, and the browser in particular.

FIG. 1 illustrates the well known, general components of a typical computer: System hardware **100** includes one or more processors (CPUs) **110** and one or more volatile and non-volatile memory devices **112**. The system hardware will typically also include I/O cards and controllers **114** as needed to communicate with and control such input devices as a mouse (or trackball, joystick, etc.) **120** and keyboard (or speech recognizer, etc.) **122**, as well as output devices such as a display monitor **124**, which display screen **126** is viewed by the user.

System software **200** is usually installed to run on the hardware "layer" **100** and will include an operating system (OS) **220** and various drivers **222** that are used for software control of physical devices; note that the drivers **222** are typically installed in the OS itself. The GUI **224** is also often an integral component of the OS. One of the features of a GUI is that, together with supporting hardware components, it senses the position of an on-screen indicator such as a cursor **230** and relates this position to other displayed graphical devices such as icons, scroll bars, data entry fields, etc. When a user positions the cursor over a data entry field and causes some predefined activating event, such as "clicking" an appropriate button of the mouse **120**, the GUI may, for example, associate text entered via the keyboard **122** with the variable or parameter with which the selected data entry field is associated.

A user-level software layer **300**, that is, applications, are usually installed to run on the system software. There are of course countless applications and it is these programs whose operation is most visible to users. In the context of the Internet, the application most relevant is known as a browser **310**. At present, the most widely used browser is Microsoft Internet Explorer, of which there are several versions.

Defined broadly, a browser is an application that interprets and arranges elements (text, sound images, etc.) on a displayed web page so that a user can see and interact with network content. Defined a bit more specifically, a browser is software that interprets the programming language of the

Internet into the text, graphics, sounds and effects (such as blinking, scrolling, etc.) that one sees when viewing a web page.

The programming language that all widely used browsers interpret is the Hypertext Markup Language (HTML), as well as its various derivatives such as DHTML, XML, DHTML, etc. HTML is standardized by the World Wide Web Consortium (W3c) and is followed by most of the leading browsers. HTML is structured as a series of elements (commonly known as "tags"), each of which contains an instruction commanding the browser how to display, for example, images and words. Different versions of HTML support not only text, multimedia, and hyperlink features but also scripting languages. When a user, via the browser **310**, contacts a web site (for example, using a URL) and requests download of content, what is transferred over the network **700** from the site-hosting server **800** is the HTML code and data, which the browser then interprets to generate the display that the user actually sees.

A "script" is a list of commands that are embedded in the HTML code defining a web page and can be executed by or within the browser. In essence, a script is a "program within a program" and can be used to affect the actions of the browser. The scripting language most commonly used in the Microsoft Internet Explorer, for example, is "Visual Basic Scripting Edition" or "VBScript" for short. VBScript is a simplified derivative of the Visual Basic programming language and enables web authors to control the action and visual characteristics of interactive controls. The time and nature of the execution of the script is controlled by the script designer and can be called from within a web document. The script is often activated and run as a result of mouse functions, buttons, or other user actions.

All of the hardware and software features and concepts described above are very common and well known. What is also common, but less noticed by users, is that conventional graphical, text-input devices generated by browsers have limited flexibility. In particular, because of the nature of existing HTML tags, the width of the displayed input field of a device such as a "pull-down menu" or "selection box" is determined by the widest possible entry, or a predefined width set by the programmer. This limitation is illustrated in FIG. 1, which shows a very simple example of a web page, with a logo **400** and data entry fields for a customer number, the date, and a state/province. This might for example be the input screen for a company that wants tracking information for all shipments of goods to a given state/province on a certain date for a given customer.

Where there is a large number of different customers, it would be impractical for the browser to display every possible customer number. Assume therefore for the sake of this example that the user has obtained the customer number before entering the illustrated input screen; for example, he may have looked up the number using a previously displayed web page. To enter the customer number the user positions the cursor **230** to point within a corresponding data entry field **410**, clicks on a mouse button to select or "activate" this field, and, using the keyboard **122**, enters the desired customer number. By pressing the "Enter" key on the keyboard, by double-clicking the mouse button, by clicking the mouse button outside the data field **410**, and/or by taking any similar field predefined action, the user indicates that he has completed his entry for the customer number. The browser then associates the entry with the corresponding parameter, which is defined as part of the HTML code. This is the operation of a standard HTML <INPUT> element.

The user may enter the date similarly, for example, selecting the Date data field **420** and typing in a date. Alternatively, it is common to display a small calendar icon **422** next to a date input field. When the user clicks on this icon, some portion of a calendar is displayed and the user can select a desired date by “clicking” on its portion of the displayed calendar. Assuming a monthly calendar is displayed, then each day would be a data selection field of its own, although no input is permitted other than selecting it with, for example, a mouse click or keyboard indication such as “Enter.” Once a date is selected or typed in, then it is similarly associated with the corresponding HTML parameter.

The example shown in FIG. 1 also includes input fields **420**, **424** and buttons **422**, **426** for the customer’s country and state/province, respectively. Consider now the State/Province data entry field **430**. In the illustrated case, the user is to select this entry using a class of display devices known as a drop-down select list. Here, when the browser detects that the user has selected the data field, or an associated activating button **432**, the browser displays a drop-down list **434** of possible choices. This list is displayed just below the data entry field **430**; although this is done to make the selection more intuitive, it is also a requirement of the HTML <SELECT> element.

Assume now that the possible country entries the USA, Canada **13**, Germany **16**, Switzerland **26** or Sweden **21**, which are further divided into their respective states, provinces/territories, Bundesländer, cantons and län. In other words, there are (as of 2002), $50+13+16+26+21=126$ choices the user might make. The question then becomes: How many characters wide should the data field **430** be displayed? The answer in all conventional browsers is: At least as many characters wide as the number of characters in the largest possible entry. In this example, this would mean that the data entry field **430**, as well as the drop-down list **434**, would need to be at least **22** characters wide in order to accommodate the possibility that someone would need to enter the German state of Mecklenburg-Vorpommern or the Swiss canton of Appenzell Ausserrhoden. This would be so even though the widest entries for the USA, Canada or Sweden would be only **14**, **21**, and **15** characters wide, respectively, in order to accommodate North or South Carolina, Northwest Territories, and Västra Götaland.

This limitation has at least one significant drawback, the common solution to which creates different drawbacks of its own: As FIG. 1 illustrates, although the left edges of the data entry fields **410**, **420**, **424**, and **430** are aligned, the right edges usually will not be, unless all of these fields are wastefully made 22 characters wide as well. Misalignment of displayed input fields is not always merely a cosmetic disadvantage, although this disadvantage may in fact be significant enough on web pages that need to appear professional and be easy to use: If the page designer wishes to include many more displayed elements on the displayed page, for example, there may not be enough space without reducing font sizes below a legible minimum. In short, the inflexibility of existing browser-based, multiple-choice graphical input devices such as pull-down menus reduces the layout freedom of the web page designer.

One common attempt to avoid the drawbacks of an overly wide input field is to require the user to select from a list of abbreviations. In many cases, there are no standardized, widely recognized abbreviations, in which case users must be forced to learn them or look them up. This is of course often inconvenient. Even where there are standardized abbreviations, however, this solution is not necessarily easy

to use since it still presupposes that all users know them. Continuing with the example shown in FIG. 1 and described above, it would be possible to list only the standard two-letter abbreviations for the possible states, provinces, etc., and countries. Not all European users would know off hand, however, that Mississippi and Missouri are abbreviated MS and MO, although this is arguably more likely than that the average American user would know that Sweden and Spain are abbreviated SE and ES, respectively.

The example shown in FIG. 1 is a mild illustration of the problem. One actual web site run by a major U.S. airline for on-line reservations had a “State” field for the user’s address. In addition to the expected 50 entries, for which a field width of 14 characters would suffice, there was also a choice “Armed Forces Americas (except Canada)”. The corresponding data entry field was therefore 37 characters wide (in fact, more, presumably to leave open the possibility of even longer possible entries), although the number of users requiring the long, 37-character entry is almost certainly miniscule compared to the number of users from the more compactly named States Iowa, Ohio, and Utah.

Another known method of reducing the necessary width of a data input field is for the possible entries to be truncated. This solution presents problems similar to those associated with abbreviations, including the drawback that an essential part of the information to be displayed may not be displayed at all.

The limitations of existing pull-down menus arise from the definitions of the HTML (for example) code with which they are created. Typically, a drop-down list is created in HTML using a <SELECT> element, which is an element that defines a set of options associated with an HTML label. Although the browser typically generates the drop-down menu and its other associated display fields (such as the button and selection box) when the page loads, it keeps these display objects invisible until the appropriate time. For example, if a user has scrolled down so far on the display screen that the menu should not be visible, then the browser ensures that it is not. The menu, its associated selection box, and the (optional) button may be made invisible or visible independently.

Associated with each input field, including those associated with <SELECT> elements, are predefined “focus events,” which are user actions that indicate to the browser that the user wishes to enter data into that field or make a selection via the list. Typical focus events include the user “clicking” on the displayed data entry field, tabbing into the field, and/or clicking on a button next to the field. When an event handler within the browser senses a focus event associated with a drop-down menu, the browser will turn on the visibility of the corresponding selection box and position it correctly, normally just under the data entry field that is to be “filled in.”

The possible selections for a pull-down menu may be determined in different ways. Note that a selection box can be considered as an array, with each row containing one possible value. There are different ways in which to populate this array with possible values. In the simplest case, all selections are included directly in the definition of the <SELECT> element and are transferred to the browser along with other HTML code used to generate the current web page. Another possibility is that the browser, upon encountering the <SELECT> element, contacts the server sending the page and downloads the possible selections, which are then stored in the array. In either case, existing browsers set the width of the data entry field of the <SELECT> element

to at least the width needed to fully display the widest selection in the array; this width may also be fixed by the programmer.

One illustration of this technique is found in web sites developed by the EXportFILE company of Reno, Nev. In the pages displayed using EXportFILE technology, a drop-down menu may be wider than the input field with which it is associated. This allows users to see all possible entries full-width while not requiring the displayed data entry field itself to be set at the maximum width. The main drawback of the EXportFILE solution, however, is that the maximum width of the drop-down selection list is predetermined in the definition of the corresponding <SELECT> element and is the same for all users. In other words, the width of the drop-down selection list, as well as all possible selections, are set and are included in the HTML code first transferred into the browser, regardless of the user. If one user has a single, very wide possible entry in a selection list, then the selection list displayed for all users will be wide enough to accommodate the one user's one wide entry. Although the selection list will disappear from the display as soon as a selection is made or is otherwise inactivated, it would still be advantageous not to display selection list any wider than necessary in order to obscure as little of the rest of the display as possible.

The EXportFILE solution may be adequate in cases where all users are to be presented with the same choices in each selection box. This is not always so, however. One example of a situation in which the one-size-fits-all approach does not work well would be where a company's employees go to customer sites for different reasons but all submit trip reports via the Internet. The different reasons that a technician might go to the site will probably be very different from the reasons that a sales representative would visit. A contract employee placed at the customer's site would have still different tasks to report back. If the on-screen reporting form includes an input field such as a task code, then the possible codes would be very different for the different reporters. Moreover, the list of possibilities may change with time. Ideally, the choices presented to users could be easily updated and would correspond to the type of services each performs; this is not possible where the HTML code and data that control the input

It is also possible to condition the width of one selection list on an entry into a different input field. Returning to the example of FIG. 1, if a user enters "USA" as the country, the browser could automatically restrict the drop-down selection list to U.S. states, so that a selection list 14 characters wide would suffice. Using existing browser technology, this is, however, practical only where the possible selections are known in advance and can be included in the definition of the corresponding <SELECT> element.

There are usually two different ways to navigate through the list of choices in a drop-down menu. Especially where only a portion of the total number of choices is displayed in the drop-down field at any one time, the first is to use the mouse (or equivalent) and cursor to move a scroll bar 436 until the desired entry becomes visible, and then select it in the normal manner. Another method, which may be combined with the first, is by using so-called "key press events." As its name implies, these events are triggered by the user performing some action on the keyboard, which the browser then converts into an appropriate action for the menu. For example, if a user presses the "Up" or "Down" arrow key when the selection box is active, the browser will highlight the entry just above or below the previous selection, respectively, and may scroll the display as needed, for example,

where there are more possible choices than can fit in the displayed selection box. Pressing the "Enter" key or double-clicking with a mouse, however, is usually interpreted as meaning that the user has selected the currently highlighted array element for entry into the data field; the browser then typically once again renders the selection box invisible.

Another common and convenient type of key press event is where the user presses one of the alphanumeric and/or symbolic keys of the keyboard, for example, in order to input the first character of the desired selection. The browser then automatically scrolls to the first selection beginning with that character. For example, if the user wishes to select "Maryland" from a drop-down list of states, then he can type "M." Assuming that the entries are ordered alphabetically, the browser will scroll the drop-down selection box to the entry for "Maine," which it will typically also highlight. "Maryland" will then be visible as the next entry and can be selected with the mouse as usual, or by pressing the "Down" key followed by the "Enter" key.

The problem with existing graphical input devices is that they do not make it convenient and intuitive to go to the "Maryland" selection directly using the alphanumeric or symbolic keys alone: In many browser-based environments, if the user were to type "M" and then "A", the browser would first scroll to and highlight "Maine," but then would immediately scroll back up to the top of the list and highlight "Alabama," leaving the user no better off than when he started.

In some other browser environments, selections are grouped according to first characters and users can cycle through each group by repeatedly entering this first character. Continuing with the example above, the first time the user enters "M," "Maine" is highlighted, since it's the first state in alphabetical order that begins with the letter "M." Repeatedly entering "M" causes the browser to highlight, in turn, Maryland, Massachusetts, Michigan, Minnesota, Mississippi, Missouri, and Montana, whereupon it returns to highlighting Maine. In order to select "Montana" using the keyboard alone, the user would therefore have to enter "MMMMMMMMMM"; "MMMMMMMMMM" would not be enough "Ms" and "MMMMMMMMMMMM" would be too many. In other words, the user would need more keystrokes (eight "Ms") to select Montana than if he had simply typed the entire name "Montana" (seven keystrokes). This is both inefficient and non-intuitive.

The problem of inefficient and non-intuitive selection of entries from a drop-down list is compounded where the entries are ordered as items, sub-items, sub-sub-items, and so on, that is, hierarchically. For example, assume that a list is to include the various job functions in a large corporation. An alphabetically ordered list would in most cases be impractical. Rather, it would in most cases be better to organize the job functions by division, group, department, task, etc. For example, the administrative assistant in the patent department might be classified as "Corporate:Legal: Intellectual_Property:Patent:Admin_Assistant." Obviously, using any of the prior art selection methods described above would be far too cumbersome.

What is needed is therefore a graphical input device similar to a selection list that eliminates the need to size displayed entry fields to the largest possible entry, while still allowing the user to see full-width, complete selections. The width of the selection list itself should be flexible and not require all users to view a list dimensioned to accommodate the widest possible entry of any user. Preferably, the input device would also allow for easier direct keyboard selection of listed items, even where the items are arranged hierar-

chically. This invention provides an input device that has either, or both of these advantageous features.

SUMMARY OF THE INVENTION

The invention provides a data input method according to which a graphical input device is generated and displayed on a display screen. The graphical input device is associated with a user-selectable parameter and has a displayed data entry field of a first display width. A set of user-dependent choices is associated with the graphical input device. When user selection of the graphical input device is sensed, a list of user-dependent choices is displayed on the screen. The list has a second display width. Selection by the user of any of the user-dependent choices is then sensed, whereupon at least a portion of the selected user-dependent choice is displayed in the data entry field and the user-selectable parameter is set to the selected user-dependent choice. The second display width is chosen as a function of display widths of the user-dependent choices, and the second display width may be greater than the first display width.

In the preferred embodiment of the invention, code for controlling the display on the display screen is downloaded from a server into a local computer. The downloaded code, which is preferably in a mark-up language, is then executing using a browser, which generates the graphical input device by executing a scripting subroutine that is embedded within the downloaded code. The mark-up language is preferably a version or derivative of HTML.

The graphical input device is preferably generated as a non-menu, text-input graphic device but that appears to the user to be a drop-down menu.

According to yet another aspect of the invention, at least first and second key press events (KPE) are associated with the list. The first KPE indicates completion of user selection of one of the user-dependent choices, and the second KPE indicates user entry of any of the characters. Upon sensing any first KPE, the list is made invisible on the display screen. A first portion of the non-HTML script is then executed to assign a currently selected one of the user-dependent choices to be the value of the user-selectable parameter. Upon sensing a first occurrence of any second KPE, a second portion of the non-HTML script is executed, which searches and marks for the user a first one of the user-dependent choices whose first character matches the user-entered character constituting the sensed second KPE.

The following procedure is then repeated as long as second KPEs are sensed, and until any first KPE is sensed: Upon sensing an n'th occurrence of any second KPE, the first one of the data entries in the list whose characters match the first through n'th user-entered characters, constituting the first through n'th occurrence of second KPEs, is searched and marked. Searching list entries is preferably done left-justified but beginning to the right of a delimiting character.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates the general configuration of a computer, the general operation of a browser, and a variety of conventional graphical input devices as found in the prior art.

FIGS. 2A–2C illustrate the general appearance and operation of a graphical input device according to the invention, in use with an example of a data input screen.

FIG. 3 illustrates certain data structures and mechanisms implemented within a server configured to operate according to the invention.

DETAILED DESCRIPTION

The invention provides a new graphical input device for allowing a user to select from one or more entries for input into an on-screen data entry field. The primary anticipated use of this new graphical input device is in the context of data input via a browser, which is to be sent over a network such as the Internet. The techniques that enable this preferred browser-based input device are also applicable, however, to other contexts, such as in spreadsheets and similar applications, in which data is input into cells (which form data entry fields) and where the cell width would otherwise need to be at least as great as the widest possible selectable entry.

The graphical input device according to the invention resembles a drop-down menu with a selection box, but the preferred embodiment of the invention has either, and preferably both of two unique characteristics. First, the displayed width of the data entry field need not be the same as the width of the longest possible selection, but rather is sized automatically according to the selections available to the current user. Second, the graphical device according to the invention preferably allows the user to make multiple alphanumeric and/or symbolic keyboard entries to more quickly find entry choices based on more than just the initial character. These features are described further below.

FIGS. 2A–2C illustrate one hypothetical situation in which an employee submits browser-based reports over the Internet about work done at a customer site. FIG. 3 illustrates certain components and mechanisms included in the server 800 that enable it to communicate with a user system 900 so as to implement the features of the invention described below. The invention may be used, however, in any situation where a user is to enter data on-screen, and in particular where at least one data entry is to be made by selecting a choice from a selection menu. In the browser-generated display screen 126 of FIGS. 2A–2C, several input devices are displayed. The other features of a general computer as shown in FIG. 1 are assumed to be present in the invention. Because they are well known, however, and for the sake of simplicity and clarity, they are not reproduced in FIGS. 2A–2C or 3.

Assume first that the employee has identified herself to the server 800 in which the invention is implemented using a browser and any conventional log-on or standard input screen. An employee such as “Rose Beck” will thus log into the server 800 by giving either her name, employee ID number, and typically also a password, and she will usually also indicate which online data entry form (such as “Customer Visit Report”) she needs. Using a conventional software module 810 for user ID and verification, the server can then identify and authenticate the user as “Rose Beck. A similar module 820 may be used to identify the desired form, for example, the illustrated “Customer Visit Report,” based on the user’s input. These are of course well known techniques and mechanisms, which enable the server to retrieve from a data base 830 the parameters 832 specific to the user, as well as code and data 834 defining the various available forms.

In FIGS. 2A–2C, in addition to showing “Rose Beck” as the reporting employee in a field 500, a conventional date entry field 504 is shown (along with a calendar icon 505), as well as data input fields 506–508 and 511 for the customer’s name, address, city, state/country and ZIP or postal code, respectively. Input fields 502, 509 and associated drop-down buttons 503, 510 are also shown for input of the type of service/task performed at the customer site and for the

customer's state/country, respectively. Although the graphical devices **502**, **503** (and possibly even **509**, **510**) appear to the user to be conventional devices in which a set of choices is displayed for the user as a drop-down selection list, they are assumed here to be implemented according to the invention. Of course, any number of graphical input devices according to the invention may be included on any given screen presentation.

Now consider an input field, such as **502**, for which a drop-down selection list is to be generated, and for which the possible choices may vary according to the user. For each such field, the set of possible choices can be considered to be a user-specific parameter that can be stored, for example, in the data base region **832**. According to the invention, a form-customization module **840** in the server retrieves the list of possible choices for the input field **520** that are to be presented to the current user, for example, from the data base region **834**. The form-customization module **840** may use any known mechanism, for example, a look-up table, to decide which set of choices to retrieve for a given user and input field on the currently chosen form.

In the preferred embodiment, the form that the user has requested is defined for the browser in HTML or one of its derivatives (such as dynamic HTML). As is well known, the HTML element known as a <DIV> defines a generic container, that is, a data space, with which alphanumeric and/or symbolic characters as well as functional parameters may be associated. The <DIV> element may contain almost any HTML element, for example, SELECT lists, tables, etc., and even other <DIVs>. Each <DIV> includes an identification (ID) parameter that can be referenced by scripting in order to change its value, font, etc.

According to the invention, the form-customization module **840** retrieves the possible choices associated with the data input field **502** (for example) and requesting user, and then populates a "container" <DIV> with "inner" <DIV> elements, each of which forms a row corresponding to a choice (one row per choice being presented to the user). The container <DIV> element is then downloaded to the user's system **900**, in particular, its browser, along with the rest of the HTML code and data that define the requested form.

The choices presented for a data input field such as "Service Type" shown in FIGS. 2A-2C may be individualized, or may be grouped into user categories, or both. For example, the Service Type choices downloaded to users who are identified (via their names, employee ID numbers, etc., depending on what information is used to identify and/or authenticate the user) as field technicians might be "Installation," "Scheduled Maintenance," "Warranty Service," "Non-Warranty Service," "Upgrading," and "Field Testing," with respective job codes CODE 01-06, whereas sales representatives might need choices such as "Existing Customer," "New Customer," "Trade Show Contact," and so on. One advantage of storing the choices in the server **800** and downloading them to the browser at the time the form is requested, along with the other HTML code, is that the choices can be easily and centrally updated as needed.

Assume now that the user selects the Service Type data entry field **502**, for example, by positioning the cursor **230** in the field **502** and clicking once. In other words, assume the user performs any predefined focus event associated with the data entry field **502**, such as tabbing into the field, clicking in it, or clicking on the button **503**. According to the invention, a software "scripting" module, which may be implemented using known programming techniques, is included in the HTML code that defines the current page and that is sent from the server **800** on which the page is hosted.

Where the browser is Microsoft Internet Explorer, the scripting will typically be VBScript.

When the browser receives the HTML code for the requested page, it also receives a list of valid options for each selection field, **502** and **509**. The lists are received as delimited text strings, which are converted into arrays by the scripting module using known techniques. The scripting module then builds the corresponding selection lists, such as list **520** for field **502**. It does this by creating inner <DIV> elements (one for each item in each array) within the container <DIV> associated with each field, **502** and **509**. These inner <DIV> elements become the options list associated with the field. The list is kept hidden from the user until the field, **502** or **509**, is activated by tabbing into it, clicking on it, clicking on the button **503**, **510** associated with it.

When creating the selection list **520**, the scripting also evaluates the width of each choice that is to be included and determines the minimum width needed to allow full-width display of all the choices. Upon sensing ("trapping") any focus event for the input field **502**, the scripting makes visible the selection list **520**, which displays the possible choices for the user, and positions it preferably as if it were a standard drop-down list below the input field **502**. The displayed width of the selection list **520** is at least the minimum width needed to display all the choices full-width unless this width exceeds the page width, in which case a horizontal scroll bar is preferably added in order to keep the select box width within the page.

Note in particular two features of the selection list **520** according to the invention: First, the selection list **520** may be wider than the data entry field **502**. Second, the width of the selection list **520** is not fixed and the same for all users, but rather will depend on, that is, be customized for, the choices to be included in the list **520** downloaded for the current user (or user type). This is illustrated in FIG. 2B, in which the selection box **520** is shown as including the choices:

```
CODE 01:Installation
CODE 02:Scheduled Maintenance
CODE 03:Warranty Service
CODE 04:Non-Warranty Service
CODE 05:Upgrading
CODE 06:Field Testing
```

Note that, whereas "Upgrading" requires an entry field only 17 characters wide for full display, the longest choice would require a display field 29 characters wide (including spaces and punctuation marks). Using conventional technology, either the data entry field **502** itself would need to be displayed 29 characters wide, which would in most cases rule out not only alignment with fields above and below it, but also any data entry field at all to either side, or all users who wish to enter data via the same screen would be shown a selection list 29 characters wide.

According to the invention, however, the data entry field **502** may be kept at any desired width, whereas the selection list **520** is allowed to be as wide as necessary to accommodate all the entries. Of course, entries even in the selection list **520** may be abbreviated or truncated as desired; this will be up to discretion of the page designer and will not be a requirement imposed by HTML syntax.

Once the selection list **520** is visible, the user may select an entry in any conventional manner, for example, by positioning the cursor over it, whereupon the scripting module preferably, as in conventional pull-down menus, highlights this entry. In FIG. 2B, for example, the user has selected "CODE 04:Non-Warranty Service." If the list of

choices is so long that all choices cannot conveniently be displayed all at once in the selection list of the chosen height, then other conventional graphical devices such as scroll bars may be included.

When the selection list **520** is displayed (made visible) by the scripting module, the selection list **520** may totally or partially cover other displayed elements such as graphics, text, or other data entry boxes. For example, in FIG. 2B, the selection list **520** covers the Date field **504**, as well as most of the text to the left of entry fields for State/Country, and ZIP/Postal Code. This will not usually inconvenience or confuse the user, however, because he will have taken actions indicating that he is at present interested in the field **502**. The scripting module in the HTML code of the page is preferably coded so that it interprets an event such as clicking outside the selection list **520** to indicate that the list should be closed. The user will then once again be able to view all the entry fields and other display elements.

Once the user chooses an entry from the selection list **520**, and performs any event (such as double clicking, pressing the "Enter" key, etc.) indicating acceptance of the chosen entry, then the scripting module inserts the chosen entry into the data entry field **502**, for example, by inserting at least a portion of the selection into the <INPUT> element defining the field. That only a portion may be inserted is because the selected entry may be too wide to fit in the displayed field. FIG. 2C illustrates how the choice "CODE 04:Non-Warranty Service" might be displayed in the field **502**, namely, it is truncated to "CODE 04:Non-Warr" merely for the purpose of display.

Note that although the display may be truncated, the actual value internally associated with the parameter corresponding to the field will remain full-width. When the user later activates (through a focus event) the field **502**, the scripting module according to the invention preferably also traps this, and then displays the entire, full-width entry, for example, by redisplaying the selection list **520** with the entry highlighted; alternatively, the entry could be made to scroll horizontally, for example, under the control of "Left" and "Right" arrow keys.

With the invention, what the user views appears to be a conventional pull-down menu, but at times with an "extra wide" selection list. According to the invention, however, instead of a standard HTML element defining a drop-down menu's entry field, what is actually generated is an <INPUT> element, with a custom-built selection list **520**, possibly with a drop-down button (such as **503**) next to it—as far as the browser's normal routines are concerned, the graphical device is not a drop-down menu. In sum, according to the invention, the scripting module, which is embedded in and acts as a subroutine executing within the transmitted HTML code, simulates a drop-down menu, but does so by using and controlling other types of graphical input devices. It is of course possible to mix conventional, fixed-width selection lists and flexibly dimensioned selection lists according to the invention in a single browser display.

The ability to display the full-width text of drop-down choices, and to custom-size selection lists "dynamically," that is, based on the choices to be presented to a current, requesting user, is of course advantageous, since it reduces the demands on users to learn or manipulate abbreviations and truncations. "Custom-building" selection lists also eliminates irrelevant choices from the current user's display and thereby not

FIGS. 2A–2C illustrate yet another advantage, namely, the increased freedom of layout made possible by the

invention: Note that not only the left but also the right edges of the entry fields **500**, **502** are aligned, even though the text they contain may vary greatly in width. Such alignment is not required, of course, as is illustrated by the "Date" field **504** being slightly narrower than those above it. The advantage of the invention is that it allows the page designer to make many more such decisions, rather than being forced to work with field widths set by default or automatically according to the limitations of HTML.

According to another aspect of the invention, the scripting module is programmed, using known VBScript (for example) techniques, to process multiple or chained key press events rather than merely single events. Consider once again the example given above of an application in which a user is to select a state or country from a drop-down menu. To reiterate, using normal HTML menus, a user who wishes to select "Maryland" and presses "M" followed immediately by pressing "A" would cause the selection list to scroll to and highlight "Maine," then immediately scroll back to and highlight "Alabama." Alternatively, entries in the selection list are grouped by first character alone, with repeated entry of the first character causing cycling within the group.

The scripting module according to the invention implements either, and preferably both, of two routines that make keyboard-based selection easier. Note that scripting is able to intercept, that is, trap, key press events and control existing HTML-generated devices such as selection boxes.

According to the routine, key press events relating to selection boxes such as the list **520** are reinterpreted by the scripting module such that searching based on each key press event is carried out from the currently chosen (for example, highlighted) entry. Thus, if the user types "M", the scripting module will scroll to and indicate "Maine." The highlighted entry will not change when the user types "A" because "Maine" still meets the criteria that its first letter is "M" and its second letter is "A." If the user then types "R," however, the scripting module will select "Maryland," which is the first (and only) entry in the example to begin with "M" "A" and "R". Other key press events may be defined to indicate movement to the top of the selection list for example "Ctrl-Home." This feature of the scripting module may also be programmed using known techniques given this description of the invention.

Because of the speed of a browser relative to user's keystrokes, it is not necessary, although it is an alternative, to maintain a pointer in the selection list and to perform subsequent searches down from the pointer. For example, it is not necessary for the scripting to keep a pointer on "Maine" just because the user has entered "MA". Rather, the scripting preferably stores an array of all key press events that corresponding to valid selections from the list, and then searches downward and with left alignment from a root member (here, for example, Alabama—the first state in alphabetical order) until it finds an initial character match for all characters entered. Thus, when the user first enters "M", "Maine will be selected because it the first in order to begin with "M." When the user then types "A", the scripting searches downward from "Alabama" and finds "Maine," since it is also the first to begin with "MA." When the user types "MAS", however, the scripting searches downward from the top member (Alabama) and finds "Massachusetts," which is the first member to have left-aligned "MAS."

The second feature of the selection search according to the preferred embodiment of the invention is that the scripting routine, when searching a selection list, preferably searches characters left-aligned to the right of a right-most delimiter. For example, in FIG. 2B, members of the selection

list 520 are prefaced by a code (CODE 01–06), which is separated by a colon (one possible delimiter) from the corresponding descriptive text. In order to select “Upgrading” from the list using keyboard entries alone, using conventional browser technology, the user would have to type “CODE 05” before reaching “Upgrading,” assuming this entry could be reached at all, which would not be possible using the standard single-character search techniques. According to the invention, the scripting first locates the right-most delimiter (“:”) and performs character searching left-aligned after it. Accordingly, the user could select “Upgrading” simply by activating the selection list (with any focus event) and typing “U”.

There is no theoretical limit to the number of delimiters that might be included. One could imagine, for example, that the full internal definition of the “Upgrading” selection might be “Field Tech:Customer Site:CODE 04:Upgrading”. Selecting this member from the selection list 520 using keyboard entries alone would be impractical or impossible using conventional browser-based technology, but is made fast, easy, and convenient using this aspect of the invention.

Recall that the scripting included within the HTML definition of the form/page preferably evaluates the choices in each selection list (such as 520) in order to determine the minimum required display width. In order to reduce not only the needed width but also redundancy, and thereby improve ease of use, the scripting could also compare choices from the left and truncate from the display any text that occurs identically in all choices, to the left of a delimiter. For example, continuing with the example above, there might be many different codes and tasks all as sub-items of “Field Tech:Customer Site:” Assuming this text occurs in all the choices to be made available to the user, then it can be deleted from the display so that only the “CODE 04:Upgrading,” and so on, would be shown. If included at all, this feature should be included only where no ambiguity or confusion could arise from such truncation. Note that this truncation is not the same as that found in the prior art, which truncates even non-repeated text and frequently obscures parts of the choices that are essential for correct reading and interpretation.

It should now be clear that the invention provides several advantages over the prior art. Because the displayed width of data input fields is not dictated by the width of the input choices (with, however, full-width viewing of the choices in the selection list), input pages may be designed to be as graphically appealing and easy to use as existing client server software. The ability of the invention to perform intelligent matches that start with the lowest sub-items (to the right of the right-most delimiter) in selection lists arranged hierarchically not only reduces how much typing the user must do, but it also enables easy navigation of long lists.

Heretofore it has been necessary to adjust the form layout for every customer or only allow partial viewing of data (or the most frequent choice). For “wide” choices it is also usually necessary to have each data field reside on its own row, which leads to long pages that cannot be viewed all at once without scrolling up and down and that cannot conveniently and clearly be printed HTML. In contrast, the invention enables “flexible” standardization, in that the same browser-based input form can accommodate the proprietary data of different users, yet still maintaining an easy-to-read and compact display.

We claim:

1. A data input method comprising:

in a mark-up language-based browser in a user’s computer, generating and displaying on a display screen a graphical input device, the graphical input device being associated with an input parameter and having a displayed data entry field of a first display width;

associating a current set of user-selectable choices with the graphical input device, each user-selectable choice having at least one pair of information fields separated by a delimiter and each information field comprising sequentially ordered characters;

sensing user selection of the graphical input device;

upon sensing user selection of the graphical input device, displaying on the screen a list of the user-selectable choices, the list having a second display width;

by executing a script locally, within the user’s computer, automatically and dynamically determining a minimum display width necessary to make visible a widest one of the user-selectable choices and automatically setting the second display width to be at least as great as the minimum display width, whereby the second display width is a function of the choices in the current set and is expandable relative to the first display width;

associating with the graphical input device at least one acceptance action;

sensing user entry of a character sequence comprising at least one character;

sequentially and character-by-character searching of at least an initial information field for each of the user-selectable choices according to each user-entered character;

for each character sequence of at least one user-entered character that matches a corresponding character sequence in the initial information field of at least one user-selectable choice, highlighting at least one of the matching choices for the user;

if no initial information field of the user-selectable choices has a character sequence matching the user-entered character sequence, highlighting for the user at least one user-selectable choice in a different information field whose initial characters match the user-entered character sequence; and

upon sensing any acceptance action by the user, setting the input parameter of the graphical input device to the currently highlighted user-selectable choice.

2. A data input method as in claim 1, further comprising downloading code and data expressed in the mark-up language for generating and controlling a screen display, the data including the current set of user-selectable choices and the code including a portion intended to generate a downloaded drop-down menu, in which:

the script is embedded in the downloaded code locally, that is, in the user’s computer, and is provided for generating and displaying the graphical input device in place of the downloaded drop-down menu but as a non-menu, text-input graphic device having the appearance of a drop-down menu.